Aix*Marseille université
Initiative d'excellence
Année 2019-20

Domaine Sciences et Technologies
Portail René Descartes

Programmation 1 : TP 8
Code UE : S14IN2I1

Projet Astéroïde : Partie 2

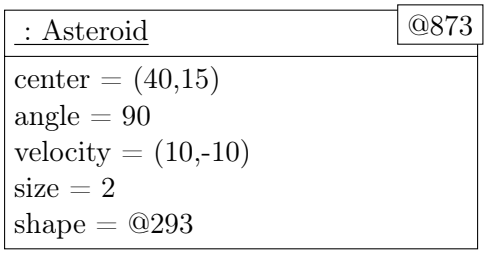
Ce TP poursuit le précédent. Il faut avoir terminer le TP7 pour pouvoir commencer ce TP.

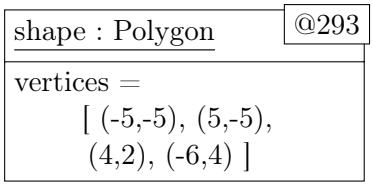
Important : pour ce TP, munissez-vous d'une feuille de papier et du crayon. Certains questions vous demanderont de faire des calculs à la main.

1 Collision point – astéroïde

La collision d'un vaisseau spatial avec un astéroïde est en général fatale. Nous souhaitons ajouter des fonctionnalités pour que le jeu se termine lorsque le vaisseau percute un astéroïde.

1. Consulter la classe `Asteroid`. Quelles propriétés d'un astéroïde permettent de déterminer son occupation de l'espace ?





2. L'astéroïde décrit par ce diagramme d'objets a ses sommets aux coordonnées $[(45, 10), (45, 20), (38, 19), (36, 9)]$. En effet, les coordonnées contenues dans la propriété `vertices` sont relatives au centre du polygone, et en supposant un angle de 0. Pour obtenir les coordonnées absolues des sommets, il faut pour chaque coordonnée relative, d'abord faire une rotation par l'angle actuel de l'astéroïde, puis une translation selon la position du centre.

   Vérifier que ce calcul produit bien les coordonnées indiquées.

3. Ajouter une méthode `public boolean contains(Vector point)` à la classe `Asteroid`, qui pour l'instant retourne toujours `false`.

4. Consulter les méthodes de la classe `Polygon`. Lesquelles de ses méthodes seront utiles pour implémenter la méthode `Asteroid.contains` ?

5. Dans un premier temps, supposons que l'astéroïde soit en position $(0, 0)$ et n'ait subi aucun mouvement ou rotation.

   Écrire la méthode `contains` sous ces restrictions.

6. En réalité, l'astéroïde a tourné et est dans une position plus générale. Il faut donc d'abord transformer sa forme en lui appliquant une rotation puis une translation.

   Modifier `Asteroid.contains` pour obtenir la version définitive de cette méthode.

7. Afin de s'assurer que l'implémentation est correcte, nous souhaitons tester la méthode `contains`, en utilisant les outils de Java.

   Créer un répertoire `test` en racine du projet. Cliquez droit sur ce répertoire, sélectionnez `Mark Directory as... > Test Sources Root`. Ceci a pour effet d'indiquer à l'IDE que ce répertoire contiendra les fichiers de tests.

8. Retourner dans la classe `Asteroid`, et positionner le curseur sur le nom de la classe, à la ligne `public class Asteroid`. Avec le raccourci clavier `Alt+Enter`, obtenir un menu dans lequel sélectionner l'option `Create Test`.

9. Dans la boite de dialogue, sélectionner `JUnit 4` dans le champ `Testing Library`. Cocher en bas la ligne contenant la méthode `contains`. Puis valider (`OK`).

   Vous venez de créer une classe `AsteroidTest` dans un sous-répertoire de `test`, avec une méthode vide `contains`. Vous allez pouvoir remplir cette méthode avec des instructions permettant de tester `Asteroid.contains`.

10. Pour tester la méthode `Asteroid.contains`, ajouter les instructions suivantes :

    (a) la création d'un polygone :

    ```
    Polygon shape =
      new Polygon(List.of(
        new Vector(-5,-5),
        new Vector(5,-5),
        new Vector(4,2),
        new Vector(-6,4)
      ));
    ```

    (b) la création d'un astéroïde `asteroid`, de forme `shape`, de centre $(40, 15)$, de vélocité nulle, de vélocité angulaire $90°s^{-1}$, et de taille 1.

    (c) la mise à jour de l'astéroïde après 1 seconde (ce qui a pour effet de le tourner de 90°).

    (d) Deux tests utilisant la méthode `contains` :

    ```
    assertTrue(asteroid.contains(new Vector(40,15));
    assertFalse(asteroid.contains(new Vector(20,45));
    ```

    Le premier test vérifie que le point $(40, 15)$ est dans l'astéroïde, le deuxième test vérifie que le point $(20, 45)$ n'est pas dans l'astéroïde.

11. Démarrer le test, en cliquant sur la flèche verte dans la marge du test.

    Un rapport s'affichera rapidement : si c'est vert c'est bon, si c'est rouge des erreurs sont détectées et il vous faut alors corriger votre programme.

12. Ajouter d'autres points au test, dans et en dehors de l'astéroïde, pour bien s'assurer que les calculs sont corrects.

    Ne pas choisir de points sur le bord de l'astéroïde : les imprécisions de calcul rendent le résultat de `contains` incertain pour ces points.

13. Ajouter une deuxième méthode de test, utilisant un autre polygone et d'autres points.

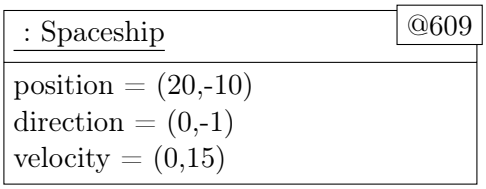
2 Collision vaisseau spatial – astéroïde

On se place maintenant dans la classe `Spaceship`.

1. Ajouter une méthode publique `collides` dans la classe `Spaceship` dont le but est de tester si le vaisseau percute un astéroïde.

   Quels sont les paramètres de cette méthode ? Quel est le type de retour de cette méthode ? Pour l'instant, faites que cette méthode retourne toujours la même valeur.

2. Implémenter la méthode, de sorte qu'il y ait collision si le centre du vaisseau (représenté par la propriété `position`) est contenu dans l'astéroïde.
3. Considérer seulement le centre n'est pas très pertinent. La classe `Spaceship` contient une liste de points de contact : ce sont des points sur le bord du vaisseau spatial. Nous supposerons que le vaisseau entre en contact avec un astéroïde si un de ces points de contact est contenu dans l'astéroïde. Les coordonnées de ces points de contacts sont données en supposant le centre du vaisseau au point $(0, 0)$ et sa direction étant $(1, 0)$.



Lister les coordonnées absolues des points de contacts pour le vaisseau de ce diagramme.

Pour avoir une coordonnée absolue, il faut d'abord faire une rotation adaptée selon la direction du vaisseau, puis une translation par la position du vaisseau.

4. Compléter la méthode `Spaceship.collides`.
Utilisez une boucle `for` sur les points de contact du vaisseau. Pour chaque point calculez sa position absolue et testez si elle est dans l'astéroïde.

3 Collision vaisseau spatial – champ d'astéroïdes

Pour cette section, on se place dans la classe `Space`.

1. Ajouter une méthode `hasCollision`, déterminant si le vaisseau spatial est en collision avec au moins un astéroïde.
2. Modifier la méthode `isGameOver` pour qu'elle retourne `true` si le vaisseau spatial est en collision.
3. Vérifier que le jeu se fige lors d'une collision entre le vaisseau spatial et un astéroïde.

4 Vies et invulnérabilités

On souhaite rendre le vaisseau spatial invulnérable aux collisions en début de partie, et lui donner un petit nombre de vies : ainsi ce ne sera qu'à la troisième collision que la partie se termine. Après chaque collision, le vaisseau devient invulnérable pendant 3 secondes.

1. Ajouter une propriété au vaisseau spatial, représentant le temps restant d'invulnérabilité, en secondes.
Si cette propriété est négative, le vaisseau est vulnérable.
2. Lui ajouter aussi un prédicat *isInvulnerable*.
3. Ajouter une méthode permettant de rendre le vaisseau invulnérable pour un nombre donné de secondes.
Si le vaisseau était déjà invulnérable, son nouveau temps restant d'invulnérabilité est le maximum du temps qu'il lui restait et du nombre de secondes reçu en paramètre.
4. Modifier la méthode `update` du vaisseau, de sorte que le temps d'invulnérabilité soit respecté.
Autrement dit, le temps restant d'invulnérabilité, s'il est positif, doit diminué avec le temps.
5. Modifier la détection de collision du vaisseau : si le vaisseau est invulnérable, il n'y a pas de collision.
6. Faire en sorte que le vaisseau soit invulnérable pendant les 5 premières secondes du jeu.

7. Toujours dans la classe `SpaceShip`, ajouter une méthode `getInvulnerabilityTime()`.

8. Dans la classe `viewModel`, ajouter des méthodes `isSpaceshipInvulnerable` et `getSpaceshipInvulnerabilityTime`.

9. On souhaite pouvoir visualiser le fait que le vaisseau est invulnérable. On propose plusieurs façons :
   — faire clignoter le vaisseau,
   — afficher une ellipse autour ou en-dessous du vaisseau,
   — changer la couleur du vaisseau,
   — afficher une barre de progression représentant le temps d'invulnérabilité restant.

   Choisir un de ces effets, ou bien un autre selon votre imagination, et le réaliser, en modifiant la méthode `render(Spaceship)` de la classe `CanvasView`.

   Après vérification que votre programme fonctionne comme attendu, retourner dans la classe `Spaceship`.

10. Ajouter des *vies* au vaisseau spatial.

    Pour cela, il vous faudra ajouter une propriété sur le nombre de vies, avec un accesseur, et correctement initialisée.

11. S'il entre en collision avec un astéroïde, il perd une vie. La partie se termine lorsque le vaisseau a perdu toutes ses vies. Après chaque collision, le vaisseau bénéficie d'une période d'invulnérabilité.

    Faire les modifications nécessaires dans les méthodes de `Spaceship` et `Space`.

12. Ajouter une méthode dans la classe `ViewModel` pour obtenir le nombre restant de vies du vaisseau.

13. Modifier `CanvasView`, en ajoutant une méthode `renderLives`, de sorte que pour chaque vie restante, une image réduite du vaisseau soit affichée dans un coin du canvas.

    Utilisez la méthode `getImage` pour obtenir l'image du vaisseau, et consultez la méthode `renderSpaceShipImage` pour voir comment l'afficher.

14. Vérifier que le programme fonctionne comme attendu.